Expense Hub: The Financial Tracker

ABSTRACT

Expense Hub Management System: A Smart Financial Tracker

"The **Expense Hub Management System** is a modern digital platform designed to streamline financial tracking, budgeting, and expense management. By replacing manual bookkeeping with an automated and intuitive system, it helps individuals and businesses gain better control over their finances.

Key features include **user authentication, real-time income and expense tracking, categorized financial insights, budget management, interactive reports, and a centralized dashboard** for monitoring financial health. Users can log in to record transactions, set budget limits, analyze spending patterns, and generate insightful reports, all within an easy-to-use interface.

Developed using **React, Firestore, HTML, CSS, and JavaScript**, the system offers a fast, responsive, and interactive experience. Firestore's cloud-based architecture ensures real-time data synchronization, enhancing accuracy, security, and accessibility across devices. By digitizing financial management, the **Expense Hub Management System** promotes financial transparency, reduces errors, and improves overall budgeting efficiency."

Introduction

Expense Hub Management System: A Smart Financial Tracking Solution

- The **Expense Hub Management System** is a powerful digital solution designed to simplify financial tracking, budget management, and expense monitoring. By automating income and expense recording, budget analysis, and real-time data synchronization, the system helps users gain better financial control while ensuring data accuracy and security.
- Developed using **React, Firestore, HTML, CSS, and JavaScript**, the system delivers a dynamic and interactive user experience with real-time updates and cloud-based storage. Firestore ensures secure and efficient data handling, enabling users to track transactions seamlessly across multiple devices.
- By digitizing financial management, the **Expense Hub Management System** reduces manual errors, enhances transparency in financial records, and improves overall financial efficiency for individuals and businesses alike.

1. Core Features of the Expense Hub Management System

2.1 User Authentication & Access Control

- Secure login system for users and administrators to prevent unauthorized access.
- Authentication using **React and Firebase Authentication**.
- Role-based access control ensures users have appropriate permissions.
- **Two-factor authentication (2FA)** for enhanced security.
- **Session management** to prevent multiple unauthorized logins.

2.2 User Account & Profile Management

- Users can create an account, log in, and manage their financial data.
- Profile customization, including income sources, expense categories, and financial goals.
- Admins can oversee and manage user activities.
- Option to set up recurring transactions for automated financial tracking.

2.3 Expense & Income Tracking

- Users can **add, edit, and delete transactions** with ease.
- Transactions are categorized into **income and expenses** for better financial organization.
- **Real-time data synchronization** using Firestore ensures seamless updates.
- **Auto-categorization of expenses** based on predefined rules.
- **Multi-currency support** for users managing finances in different currencies.

2.4 Budgeting & Financial Planning

- Users can **set monthly and yearly budgets** to track spending limits.
- Alerts notify users when they **exceed budget thresholds**.
- Budget comparison with past months for better financial insights.
- AI-driven **spending pattern analysis** to suggest savings opportunities.

2.5 Reports & Analytics

- **Interactive dashboards** displaying financial trends and expense breakdowns.
- Reports include **income vs. expenses, category-wise spending, and savings progress**.
- **Data visualization using charts and graphs** for better financial insights.
- Reports can be **exported as PDFs or Excel files** for offline analysis.
- AI-powered **predictive financial insights** to help users optimize expenses.

2.6 Notifications & Alerts

- Users receive **real-time alerts** for budget overspending.
- **Push notifications for bill payment reminders** and financial milestones.
- Admin notifications for suspicious activities and system maintenance updates.
- Periodic alerts on **savings progress and financial tips**.

2.7 Secure Transactions & Data Storage

- **Firestore's cloud storage** ensures secure financial data management.
- End-to-end encryption for **transaction security**.
- **Automatic backups** prevent data loss.
- **Offline access** allows users to view recent transactions without an internet connection.

2.8 Multi-Language Support & Accessibility

- Platform supports **multiple languages** for global accessibility.
- **Dark mode** for enhanced user comfort.
- **Voice input support** for easy transaction entry.
- **Accessibility features** for visually impaired users.

2.9 Integration with Third-Party Services

- Connect with **bank APIs** for automatic transaction imports.
- Integration with **Google Sheets and accounting software**.
- **Export transaction data** for tax filing and external reports.

2.10 Admin Dashboard for System Oversight

- Admins can **monitor user activities** and system performance.
- **Manage user accounts**, flag fraudulent activities, and assist users.
- Generate system-wide reports on **user engagement and financial trends**.
- Maintain **logs and system analytics** for performance tracking.

1. Technology Stack

1.1 Frontend Technologies

- HTML – Defines the structure of web pages.
- CSS – Provides styling and layout control.
- JavaScript & React – Enables interactivity, component-based UI, and dynamic content rendering.
- Tailwind CSS (or Bootstrap) – Ensures a responsive and mobile-friendly UI.

1.2 Backend & Database Technologies

- Firestore (Firebase Database) – Manages real-time structured storage of user transactions, budgets, and financial data.
- Firebase Authentication – Handles secure user authentication and access control.
- Node.js (optional) – Can be used for server-side operations, real-time notifications, or API integrations.

2. User Interface Design

2.1 User-Friendly Navigation

- Intuitive Dashboard – Displays an overview of key financial metrics like income, expenses, and savings.
- Search & Filter Options – Users can quickly find transactions based on categories, dates, or amounts.
- Interactive Financial Reports – Charts and graphs to visualize spending habits.
- Responsive Design – Optimized for desktops, tablets, and mobile devices.
- Clear Labels & Validations – Forms and input fields include validation messages for better user experience.
- Feedback Mechanism – Users can submit feedback or report issues for continuous improvement.
- Wishlist-like Feature – Users can bookmark expenses for review or track planned purchases.

3. Security & Data Management

3.1 User Authentication

- Secure login and access control mechanisms.
- Role-based permissions ensure restricted access to financial data.
- Two-factor authentication (2FA) for enhanced security.

3.2 Data Encryption

- Firestore security rules protect sensitive financial data.
- Secure hashing techniques for passwords.
- SSL encryption for secure data transmission.

3.3 Audit Logs

- Tracks financial transactions, modifications, and user activity.
- Admin access to logs for security and fraud detection.

3.4 Automated Backups

- Automatic Firestore backups ensure data safety and recovery.
- Cloud storage integration for redundancy and disaster recovery.

2. How It Works?

2.1 Backend Functionality

1. Firestore Database

- Firestore (NoSQL) is used to store financial data, including user transactions, budgets, and reports.
- Real-time synchronization ensures instant updates across all devices.
- Firestore security rules prevent unauthorized access to sensitive financial data.

2. Authentication & Authorization

- Firebase Authentication secures user login and account management.
- Supports email/password authentication, Google Sign-In, and social logins for seamless access.
- Role-based access control (RBAC) restricts admin and user permissions.

3. Middleware & Validation

- React and Firestore validate user inputs, ensuring accurate financial tracking.
- Prevents incorrect data entries, duplicate transactions, and unauthorized changes.

4. Asynchronous Operations

- React fetches and updates financial data in real-time using Firestore listeners.
- No need for manual page refresh—transactions appear instantly.

5. RESTful API Integration (Optional)

- The system can integrate with third-party APIs for currency exchange rates, tax calculations, and financial analytics.
- Webhooks can be used for automatic notifications and transaction alerts.

6. Error Handling & Logging

- Firebase Crashlytics tracks system errors and reports potential failures.
- React error boundaries ensure smooth handling of unexpected issues.

7. Security Measures

- Firestore security rules enforce strict read/write permissions.
- Data encryption & hashing protect user credentials and financial data.
- CORS policies and OAuth tokens prevent unauthorized API access.

2.2 Search Feature for Retrieving Transactions & Expenses

1. User Input

- Users enter a keyword, date, or amount in the search bar to find specific financial records.

2. Database Query Execution

- Firestore queries filter and retrieve relevant transactions based on user input.
- Indexing ensures quick search results without performance delays.

3. Server Processing

- The system sorts results based on category, amount, or date for better visibility.

4. Data Response

- Firestore returns matching transactions in JSON format to the React frontend.

5. Data Presentation

- Transactions are displayed in a structured format, including category, date, amount, and payment method.

6. Error Handling

- If no records match, an error message suggests alternative search criteria.

3. Benefits

The Following Flow Diagram Represents the highlighting the benefits for users:

FIGURE 3: Flow Diagramatical Representation of Benefits of Finance Hub

4. Future Enhancements

To improve the Finance Tracker System, upcoming updates will focus on enhancing security, automating financial insights, and improving the user experience. These features will help users manage their finances more effectively and make informed decisions.

Potential Future Enhancements:

✓ AI-Based Expense Categorization: Implementing machine learning to automatically categorize transactions and provide spending insights.

✓ Budget Forecasting & Predictions: Using predictive analytics to estimate future expenses and income trends based on past data.

✓ Mobile App Integration: Developing a dedicated mobile application for easy access to financial reports, transactions, and budgeting tools.

✓ Multi-Currency Support: Allowing users to track expenses and income in multiple currencies with real-time exchange rate updates.

✓ Automated Bill Payments & Reminders: Integrating scheduled payments and alerts to ensure timely bill settlements.

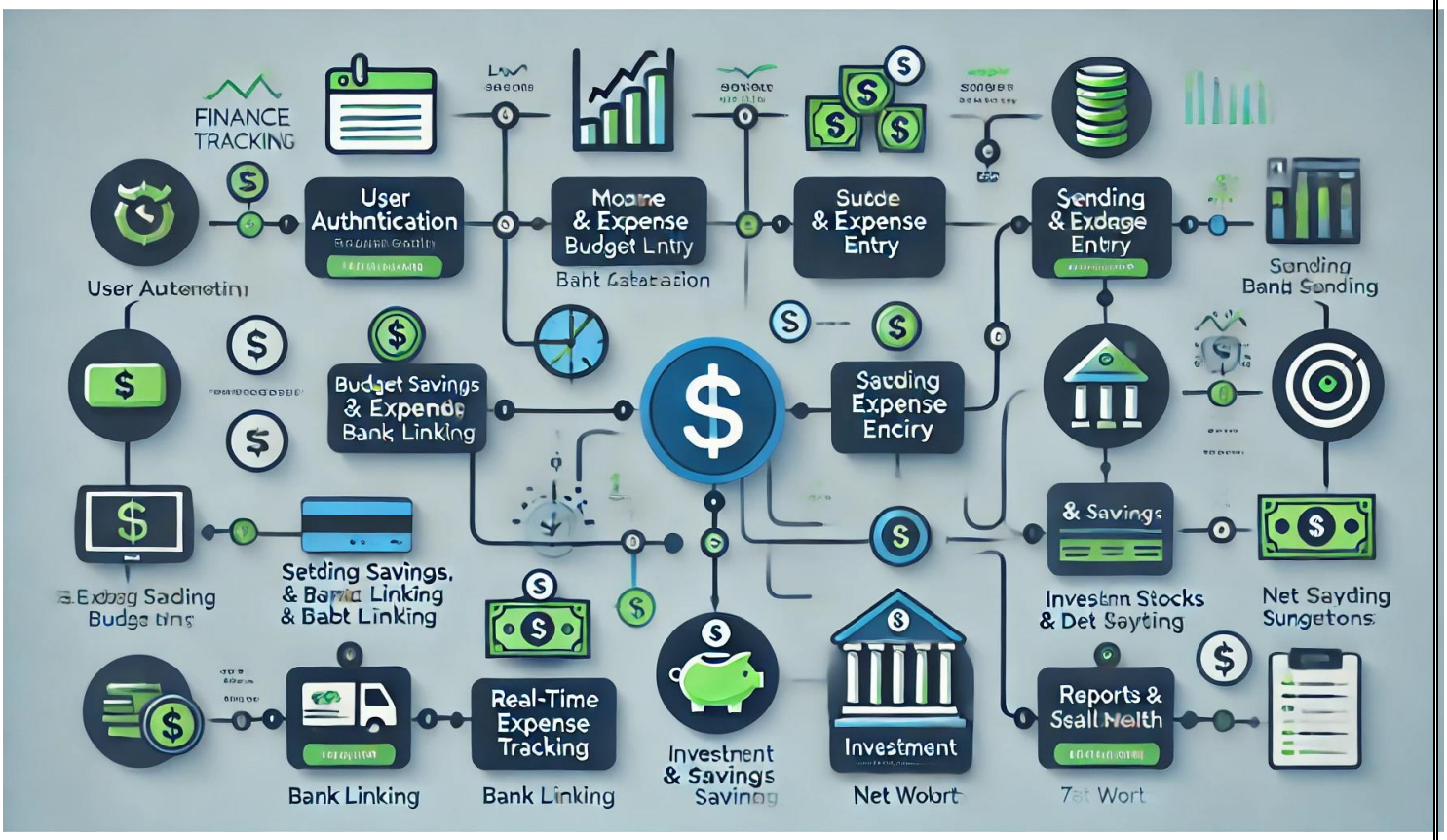
✓ Blockchain for Secure Transactions: Enhancing financial data security by leveraging blockchain technology for encrypted transaction storage.

✓ Investment Portfolio Tracking: Adding features to track stocks, mutual funds, and crypto investments for comprehensive financial management.

✓ Voice & Chatbot Assistance: Integrating AI-powered voice commands and chatbots for quick financial queries and transaction logging.

✓ Tax Calculation & Reporting: Automating tax estimations and generating tax reports based on user financial data.

✓ Dark Mode & Custom Themes: Enhancing user experience with customizable UI themes and accessibility options.



🔄 Finance Tracker Process Flow

1️⃣ User Authentication & Security 🔐

- Users log in securely using:
- Email & Password
- Two-Factor Authentication (2FA)
- Biometric Authentication (Optional)
- Ensures that only authorized users can access financial data.

2️⃣ Income & Expense Entry 💰

- Users can:
- Manually enter income and expenses

- Automate transactions by linking bank accounts
- Categorize spending (Food, Rent, Travel, etc.)

3️⃣ Budget Planning & Goal Setting 🎯

- Users set financial goals such as:
- Monthly budgets (e.g., limit on dining expenses)
- Savings targets (e.g., save $5,000 for a vacation)
- Debt repayment plans

4️⃣ Real-Time Expense Tracking 📊

- Dashboard Overview: Displays income, expenses, and savings progress.
- Daily, Weekly & Monthly Reports:
- Graphs & Pie Charts for better visualization.
- Alerts for overspending in any category.

5️⃣ Bill & Subscription Management 🧾

- Users receive reminders for:
- Utility bills (Electricity, Internet, Water, etc.)
- Loan EMI payments
- Subscription renewals (Netflix, Spotify, etc.)

6️⃣ Investment & Savings Insights 📈

- Users can:
- Track investments (Stocks, Mutual Funds, Crypto, etc.)
- Analyze interest earnings from bank savings.
- Receive AI-driven suggestions for better financial planning.

7️⃣ Reports & Financial Health Analysis 📑

- Generates:

- Spending trends over time
- Net worth calculations
- Customized tax reports for easy filing.

🔄 Automation & Smart Features

✓ AI-Powered Recommendations: Personalized financial insights.
✓ Automated Expense Categorization: Smart tagging of transactions.
✓ Multi-Currency Support: Tracks finances globally.
✓ Cloud Backup & Data Encryption: Ensures data security.

how Online Financial Tracker System is better from Offline Financial System?

Feature	🟢 Online Financial Tracker	🔴 Offline Financial Tracker System
Accessibility 🌍	Accessible anytime, anywhere via web or mobile.	Requires physical access to files or specific software.
Automation & Accuracy 🤖	Automates transaction recording, categorization, and reconciliation.	Manual data entry, increasing errors and inconsistencies.
User Authentication 🔐	Secure login with OTP, biometrics, or credentials.	Relies on local device security, more vulnerable to unauthorized access.
Real-time Updates 📊	Live sync with bank accounts, credit cards, and investments.	Requires manual updates; delays in reflecting financial status.
Backup & Security 🔄	Cloud storage ensures data safety from hardware failures.	Risk of data loss due to device failure, theft, or accidental deletion.
Payment & Integration 💳	Supports multiple payment modes and links with digital wallets.	Mostly manual entry; limited integration with digital payments.
Error & Fraud Reduction 🚨	Automation reduces errors and fraud risks.	Prone to human errors and financial mismanagement.
Data Analytics 📈	Generates real-time reports on spending, trends, and budgets.	Limited tracking; manual calculations required.
Scalability 🚀	Can handle large financial data seamlessly.	Limited to local storage and software capabilities.

4.1 Overview of HTML File

1. **Header:**
    - Contains the title of the Purchase Hub Management System.
    - Includes a navigation bar with links for users to explore products, view their cart, and manage their profile.
    - Provides a search form where users can enter product names or IDs to quickly find items.
2. **Main Content Area:**
    - Displays product listings, including images, names, prices, and availability.
    - Shows search results dynamically, updating content based on user queries.
    - Contains an interactive shopping cart section, where customers can view selected products, adjust quantities, and proceed to checkout.
3. **Footer:**
    - Provides a brief system description and user guidance.
    - Includes customer support information and contact details for inquiries.
    - Displays a feedback form link, encouraging users to submit suggestions for system improvements**.**
4. **Feedback Form Link:**
    - An anchor tag linking to a Google Form where users can provide ratings, complaints, and suggestions to improve the system.
5. **Script Integration:**
    - Links to the JavaScript file (script.js) to handle functionalities like search filtering, product selection, and cart updates.

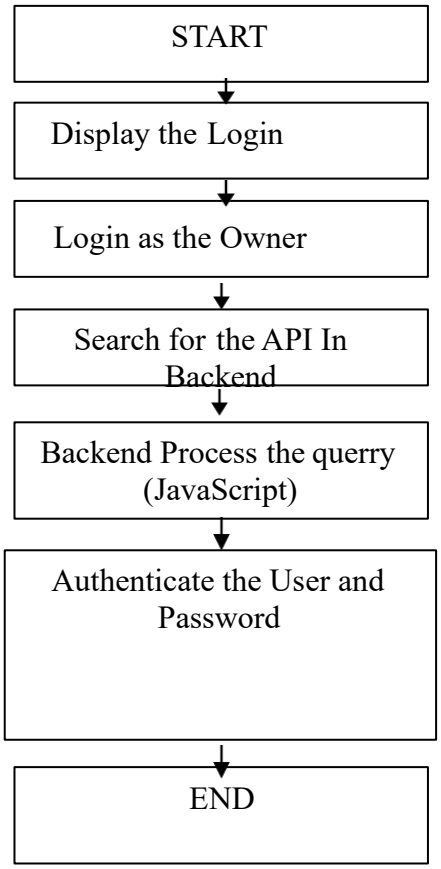


FIGURE 6: flow diagram representation of Login

Explanation of the Flowchart:

This flowchart represents the **login authentication process** for an application where an owner logs in, and the system verifies their credentials through backend processing.

1. **START**
    - The process begins.
2. **Display the Login**
    - The login screen is displayed, prompting the user to enter their credentials (username and password).
3. **Login as the Owner**
    - The user (owner) enters their credentials to access the system.
4. **Search for the API in Backend**
    - The system searches for the relevant API in the backend to verify the login credentials.
5. **Backend Processes the Query (JavaScript)** ○ The backend, which is built using JavaScript, processes the query. ○ This includes checking the username and password against stored records in the database or authentication service.
6. **Authenticate the User and Password** ○ If the credentials match, the user is authenticated and granted access.
    - If incorrect, an error message is displayed.
7. **END**
    - The process ends, either with successful authentication or denial due to incorrect credentials.

4.2 Overview of JavaScript File

This section provides an overview of the **JavaScript file** used in the **Ration Shop Management System**, detailing its core functionalities:

1. **Constants**
    - Defines constants such as API endpoints, local storage keys, and important configuration settings.
2. **DOM Elements**
    - Retrieves and manipulates necessary DOM elements, such as user input fields, buttons, tables, and notification messages.
3. **Utility Functions**
    - validateInput: Ensures that user inputs (e.g., ration card number, product quantity) are valid.
    - formatDate: Formats dates for transaction history and order logs.
    - showMessage: Displays alerts or messages to the user.
4. **Fetching Data**
    - Defines an asynchronous function fetchRationItems() to retrieve available ration stock details from the backend API.
    - Another function fetchUserData() loads user-specific details, such as ration card status and purchase history.
5. **Managing Orders**
    - placeOrder(): Handles order submission, validates stock availability, and updates the ration stock accordingly. ○ updateStock(): Reduces stock quantities upon successful order placement and updates the UI.

6. **Event Listeners** ○ Listens for form submissions, button clicks, and API responses. ○ Detects user actions such as placing an order, checking ration availability, and updating user details.
7. **Initial Data Load**
    - ○ Calls functions to load stock information, user details, and transaction history when the page loads.

How to Use the Financial Tracker Application?

1. Access the Application

- Open your web browser and navigate to the application's URL, or launch the application if installed on your device.

2. Sign Up or Login

- **Sign Up:** New users need to register with their email and password.
- **Login:** Enter your credentials to access your account.

3. Add Income Details

- Navigate to the "Add Income" section.
- Enter the source, amount, and date of income.
- The system records and categorizes the income.

4. Track Expenses

- Go to the "Add Expense" section.
- Enter expense details such as category, amount, and date.
- The system updates the expense records and categorizes them.

5. View Financial Summary

- The dashboard displays total income, expenses, and remaining balance.
- Graphs and charts help visualize financial trends.

6. Set Budget Goals

- Users can set monthly or weekly budget limits.
- Alerts notify users when spending exceeds budget limits.

7. Generate Reports

- View detailed reports on income and expenses over time.
- Export reports in PDF or Excel format for further analysis.

8. Secure Data and Logout

- All data is encrypted for security.
- Users can log out after tracking finances for security purposes.

Overview of System Architecture

1. Frontend:

- Built using **HTML, CSS, JavaScript, React ,Bootstrap** for a user-friendly interface.
- Uses **AJAX or Fetch API** for smooth data interactions.

2. Backend:

- Developed using **FireStore** for API handling.
- Implements **RESTful web services** for processing transactions.

3. Database:

- Uses **MySQL** to store:
  - User details
  - Income and expense records
  - Budget and reports
- **JPA & Hibernate** for database interactions.

4. API Integration:

- RESTful APIs for authentication, financial tracking, and reporting.
- Allows integration with third-party financial tools or banking APIs.

5. Security & Authentication:

- **JWT authentication** ensures secure user login.
- **Role-based access control (RBAC)** for different user roles.

6. Deployment:

- Can be deployed on:
  - **Cloud platforms** (AWS, Google Cloud, Azure)
  - **Spring Boot + Docker** for containerized deployment
  - **Frontend hosted on Netlify/Vercel, backend on Heroku**

Roadmap for Future Development

1. Enhanced Security & Authentication

- Implement **multi-factor authentication (MFA)**.
- Strengthen encryption methods for financial data.

2. AI-Based Expense Prediction

- Use **machine learning algorithms** to analyze spending patterns.
- Provide personalized savings suggestions.

3. Automated Expense Categorization

- Auto-classify expenses using AI-based recognition.
- Allow users to customize categorization rules.

4. Bank Account Synchronization

- Integrate with banks for **real-time transaction tracking**.

- Enable users to manage multiple accounts within the app.

5. Bill Reminders & Recurring Payments

- Users can set bill payment reminders.
- Enable **automated payment scheduling** for recurring expenses.

6. Mobile App Development

- Build a **native mobile app (Android & iOS)**.
- Provide **push notifications** for budget alerts.

7. Multi-Currency & International Support

- Support multiple currencies for global users.
- Allow currency conversion for financial analysis.

8. Advanced Data Visualization

- Implement **interactive dashboards**.
- Provide insights through **data visualization charts**.

9. Expense Sharing & Family Accounts

- Enable **shared expenses tracking** for families.
- Provide **group budgeting features** for households.

10. API for Third-Party Integration

- Provide an **open API** for fintech app integration.
- Enable **financial data export** for tax purposes.

Flow of the Financial Tracker System

1. **User Signs Up / Logs In**
    - The user registers a new account or logs into an existing one.

2. **Dashboard Overview**
    - Displays total income, expenses, savings, and financial trends.
3. **Add Income Details**
    - User inputs income sources (e.g., salary, freelance, investments) with amount and date.
4. **Add Expense Details**
    - User records expenses by category (e.g., groceries, rent, entertainment) with amount and date.
5. **Categorization & Budgeting**
    - Transactions are categorized, and users can set budgets for different expense types**.**
6. **Generate Reports & Insights**
    - System generates reports, graphs, and spending trends to analyze financial health.
7. **View & Manage Transactions**
    - Users can review, edit, or delete past income and expense entries.
8. **Savings & Goal Tracking**
    - Users set financial goals (e.g., saving for a vacation, emergency fund) and track progress.
9. **Alerts & Notifications**
    - System notifies users of budget limits, upcoming bills, or unusual spending patterns.
10. **Export & Backup Data**

- Users can export transaction history as CSV/PDF or sync data to cloud storage for backup.

4.2 Overview of CSS File:

1. **Global Styles**:
    - Sets up the font family from Google Fonts ('Poppins') and defines primary and secondary color variables.
2. **Body Styles**:
    - Defines background color, font family, and margin for the entire document body.
3. **Header Styles**:
    - Styles the header section with padding, background color, and text alignment.
4. **Search Input Styles**:
    - Styles the search input field with background color, border, border-radius, font size, and padding. Also sets placeholder color and styles the input field on focus.
5. **Main Section Styles**:
    - Sets up display properties for the main section, enabling flexbox layout and centering movie elements.
6. **Rating Styles**:
    - Styles movie ratings with background color, padding, border radius, and font weight. Includes special styles for ratings above 7 (green) and below 7 (red).
7. **Overview Styles**:
    - Styles the overview section that appears when hovering over a movie card, including background color, padding, position, and transition effects.

8. **Footer Styles**:

- Styles the footer section with text color and alignment.

9. **Container Styles**:

- Styles the container element for centering content vertically.

10. **Media Queries**:

- Provides responsive styling for smaller screens, adjusting header layout and padding.

Overall, our CSS code effectively styles the different elements of your movie search web application, providing a cohesive and visually appealing user interface.

5. TESTING

5.1 *Testing Strategies and Methodologies:*

1. **Unit Testing**:

- Write unit tests to validate individual functions and components of your JavaScript code.
- Test functions responsible for fetching movie data, processing search queries, and handling user interactions.
- Verify that functions return the expected results and handle edge cases and error conditions appropriately.

2. **Integration Testing**:

- Perform integration testing to ensure that different parts of your application interact correctly.

- Test the integration between frontend and backend components, such as sending search queries to the backend and receiving movie data for display.
- Verify that API requests are made correctly and responses are parsed and displayed accurately.

3. **End-to-End (E2E) Testing**:
   - Conduct E2E testing to validate the entire user workflow of your application.
   - Write test scripts to simulate user interactions, such as entering search queries, submitting forms, and verifying search results.
   - Use E2E testing frameworks like Cypress or Selenium to automate browser-based testing and ensure that the application behaves as expected from the user's perspective.

4. **User Acceptance Testing (UAT)**:
   - Involve real users or stakeholders in UAT to gather feedback and ensure the application meets their needs and expectations.

- Provide users with test scenarios and tasks to perform, such as searching for specific movies or providing feedback through the application.
- Gather feedback on usability, functionality, and overall user experience to identify areas for improvement.

5. **Accessibility Testing**:
   - Conduct accessibility testing to ensure your application is usable by all users, including those with disabilities.
   - Test for accessibility issues such as keyboard navigation, screen reader compatibility, and contrast ratios.
   - Use accessibility testing tools and guidelines to identify and address accessibility barriers in your application.

6. **Cross-Browser and Cross-Device Testing**:
   - Test your application across different browsers and devices to ensure compatibility and consistency.
   - Verify that the application renders correctly and functions as expected on various browsers (e.g., Chrome, Firefox, Safari) and devices (e.g., desktop, mobile, tablet).
   - Use browser testing tools or device emulators to simulate different environments and configurations.

7. **Performance Testing**:
   - Evaluate the performance of your application under different conditions, such as varying levels of traffic and load.
   - Measure response times, page load times, and resource utilization to identify performance bottlenecks and optimize application performance.
   - Use performance testing tools and techniques to simulate real-world scenarios and scale your application to handle increased traffic.

8. **Regression Testing**:

- Implement regression testing to ensure that new code changes do not introduce regressions or break existing functionality.
- Automate regression tests for critical features and functionalities to verify that they continue to work as expected after each code change.
- Run regression tests as part of your continuous integration (CI) pipeline to catch and fix issues early in the development process.

By incorporating these testing strategies into project, you can validate the functionality, performance, and usability of cine search web application and deliver a high-quality user experience to audience.

5.2 *Testing and Quality Assurance:*

1. **Bug Reports and Issues Identified**:

- Testing may uncover bugs, errors, or unexpected behavior in your application.
- Issues may be reported related to functionality, performance, accessibility, compatibility, or user experience.

2. **Test Cases Executed and Results Recorded**:

- Test cases are executed according to the testing plan, covering various scenarios and functionalities.
- Test results are recorded, documenting the outcomes of each test case, including pass/fail status and any deviations from expected behavior.

3. **Defects and Improvements Documented**:

- Defects and areas for improvement are documented, including detailed descriptions of issues, steps to reproduce, and severity levels.

- Suggestions for enhancements or optimizations may be identified based on user feedback or testing results.

4. **Regression Testing Completed**:
    - Regression testing ensures that new code changes do not introduce regressions or break existing functionality.
    - Existing features and functionalities are retested to verify that they still work as expected after code changes or updates.

5. **Performance Metrics Analyzed**:
    - Performance testing provides insights into the speed, responsiveness, and scalability of your application.
    - Performance metrics such as response times, page load times, and resource utilization are analyzed to identify areas for optimization and improvement.

6. **Accessibility Compliance Checked**:
    - Accessibility testing ensures that your application is accessible to users with disabilities and meets accessibility standards.
    - Compliance with WCAG guidelines and accessibility issues are identified, addressed, and documented.

7. **Cross-Browser and Cross-Device Compatibility Verified**:
    - Cross-browser and cross-device testing validate that your application works correctly across different browsers, devices, and screen sizes.
    - Compatibility issues are identified and resolved to ensure a consistent user experience across various platforms.

8. **User Feedback Gathered and Analyzed**:
    - User acceptance testing (UAT) collects feedback from real users or stakeholders about the application's usability, functionality, and overall experience.
    - User feedback is gathered, analyzed, and used to prioritize improvements and enhancements.
9. **Quality Assurance Sign-off**:
    - Based on the results of testing and quality assurance activities, a quality assurance sign-off may be provided.
    - The application is deemed ready for release or deployment, indicating that it meets quality standards and requirements.
10. **Continuous Improvement Initiatives Identified**:
    - Testing and quality assurance efforts may uncover areas for ongoing improvement and refinement.
    - Continuous improvement initiatives are identified to address feedback, enhance features, and optimize performance over time.

Overall, the resulting outcomes of testing and quality assurance activities contribute to the overall quality, reliability, and user satisfaction of your movie search web application, ensuring that it delivers a seamless and enjoyable experience to users.

Financial Tracker System

1. Flow of the Financial Tracker System

1. User Registration & Login

- New users sign up with personal details.
- Existing users log in with credentials.
- Two-factor authentication (2FA) for added security.

2. **Dashboard Overview**
    - Displays total income, expenses, and savings.
    - Provides charts for financial insights.
3. **Adding Transactions**
    - Users input income and expense details.
    - Categorize transactions (e.g., food, rent, utilities).
    - Attach receipts or notes for reference.
4. **Budget Management**
    - Set monthly or category-wise budgets.
    - Get alerts when nearing budget limits.
5. **Expense Tracking & Analysis**
    - View spending trends using graphs and reports.
    - Compare expenses across different months.
6. **Savings & Investment Tracking**
    - Monitor savings goals and progress.
    - Track investments (stocks, mutual funds, etc.).
7. **Bill Reminders & Recurring Payments**
    - Set reminders for upcoming bills.
    - Automate recurring payments (subscriptions, rent).
8. **Report Generation**
    - Export financial reports in PDF or Excel format.

- Yearly and monthly breakdowns of expenses and savings.

9. **Multi-User Access (if applicable)**
    - Family members or business teams can share access.
    - Role-based permissions (e.g., admin, viewer).
10. **Logout & Data Backup**

- Users can log out securely.
- Data is automatically backed up on cloud storage.

2. System Architecture

1. **Frontend**
    - Built with React.js / Angular / Vue.js.
    - Responsive UI for desktop and mobile.
2. **Backend**
    - Developed using Spring Boot (Java) or Node.js.
    - Handles API requests and authentication.
3. **Database**
    - MySQL / PostgreSQL for storing user transactions.
    - NoSQL (MongoDB) for flexible storage of logs.
4. **Security & Authentication**
    - Uses JWT (JSON Web Token) for secure login.
    - Data encryption for financial details.

5. **Third-Party API Integration**
    - Integrates with bank APIs for transaction imports.
    - Uses payment gateways for bill payments.
6. **Deployment**
    - Cloud deployment on AWS / Google Cloud.
    - Mobile app version for Android & iOS (Future Scope).

---

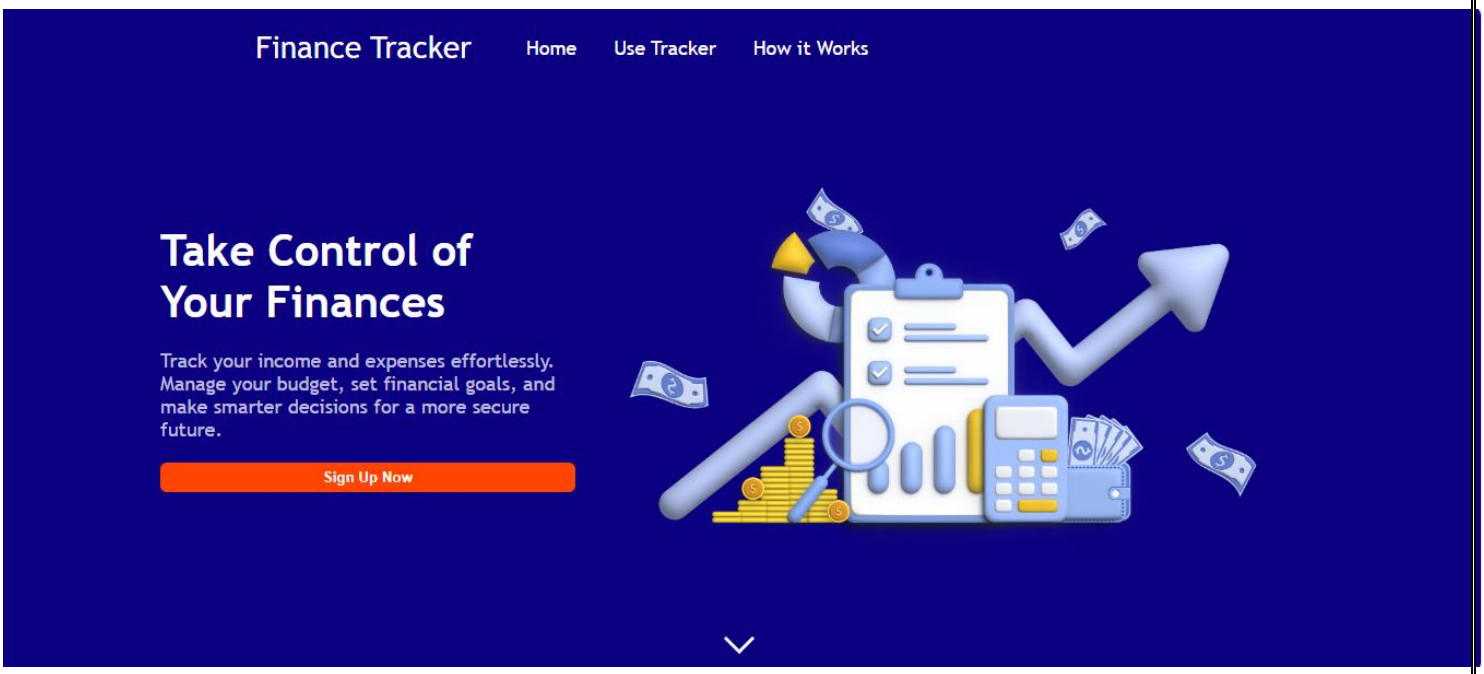
3. Roadmap for Future Enhancements

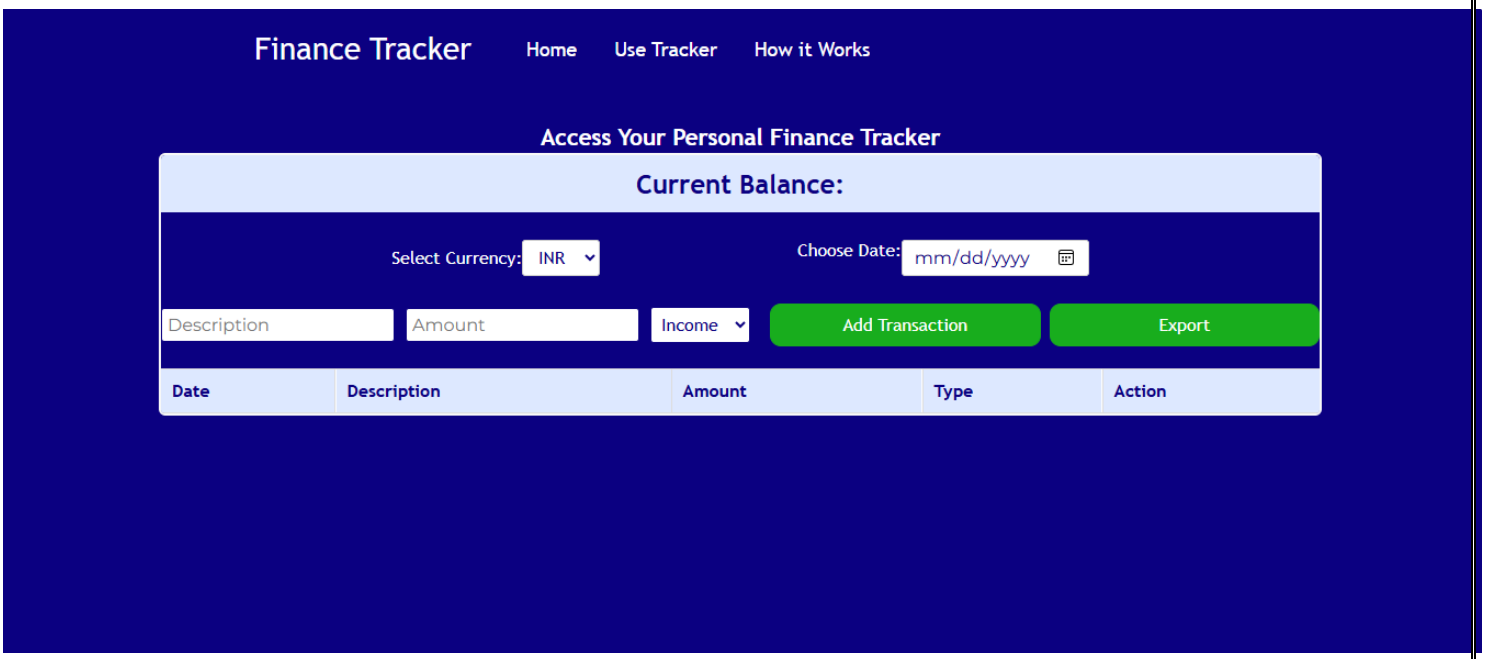
- AI-Powered Expense Predictions: Suggests spending adjustments based on trends.
- Stock & Investment Portfolio Tracking: Real-time updates on stock market holdings.
- Tax Calculation & Filing Assistance: Helps in tax planning and filing returns.
- Offline Mode: Allows data entry without internet access.
- Multi-Currency Support: Track expenses in different currencies.

ABBREVIATIONS / ACRONYMS

1. **POS** : Point of Sale
2. **PDS** : Public Distribution System
3. **FPS** : Fair Price Shop
4. **OTP** : One-Time Password
5. **DBMS** : Database Management System
6. **SKU** : Stock Keeping Unit
7. **ERP** : Enterprise Resource Planning
8. **MIS** : Management Information System
9. **GST** : Goods and Services Tax
10. **QR** : Quick Response (Code)
11. **RFID** : Radio Frequency Identification
12. **SMS** : Short Message Service
13. **UID** : Unique Identification
14. **E-KYC** : Electronic Know Your Customer
15. **NFC** : Near Field Communication
16. **BI** : Business Intelligence
17. **IoT** : Internet of Things
18. **AI** : Artificial Intelligence
19. **OTP** : One-Time Password
20. **BPL** : Below Poverty Line

Screenshots





Finance Tracker Home Use Tracker How it Works

How it Works

Experience the power of
Personal Finance Tracker

Track effortlessly
Control transactions
Customize currencies
Edit with ease
Real-time balance
Secure data protection

Sign Up Now

Financely

Sign Up on Financely

Enter your name

Enter your email

Password

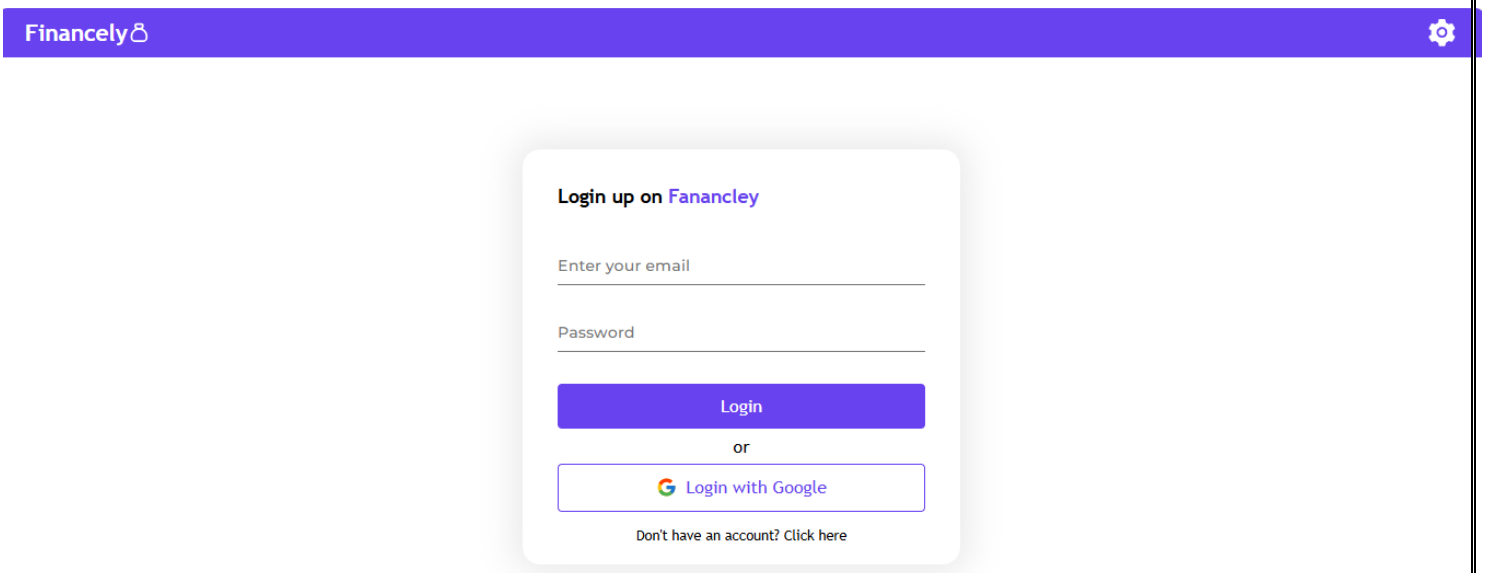
Confirm password

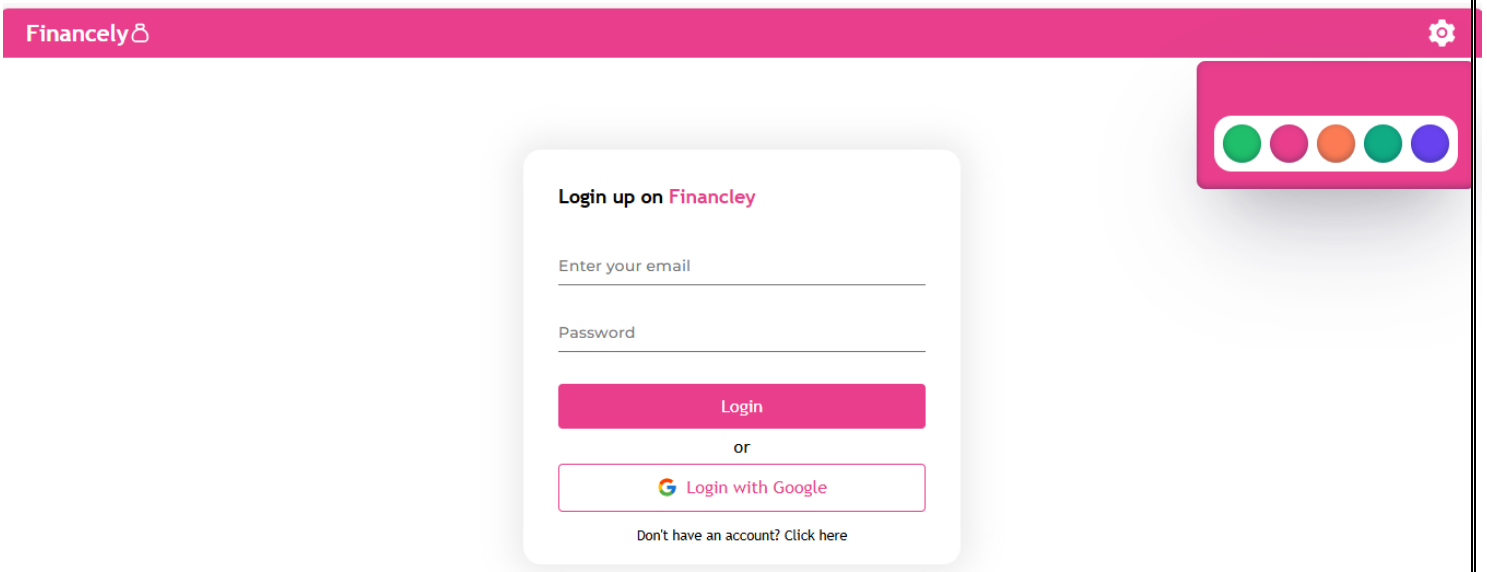
Sign up

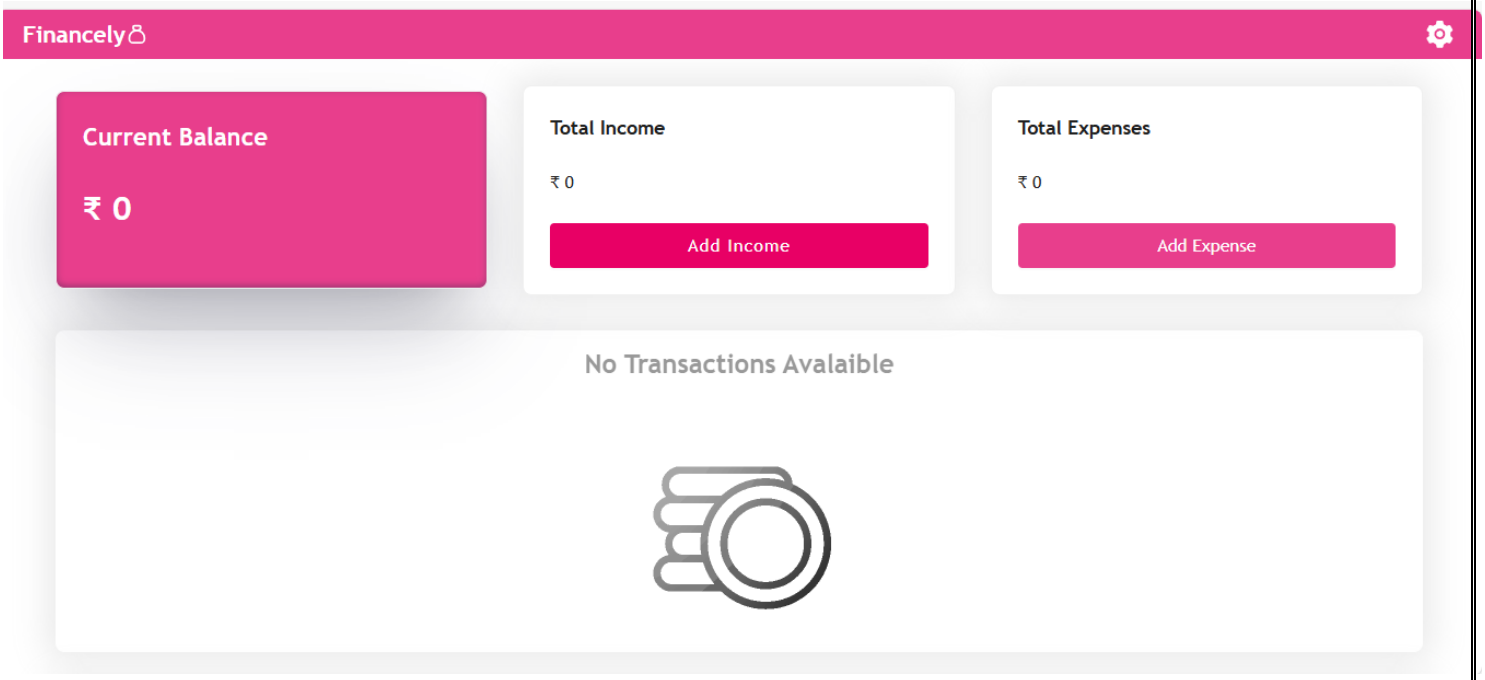
or

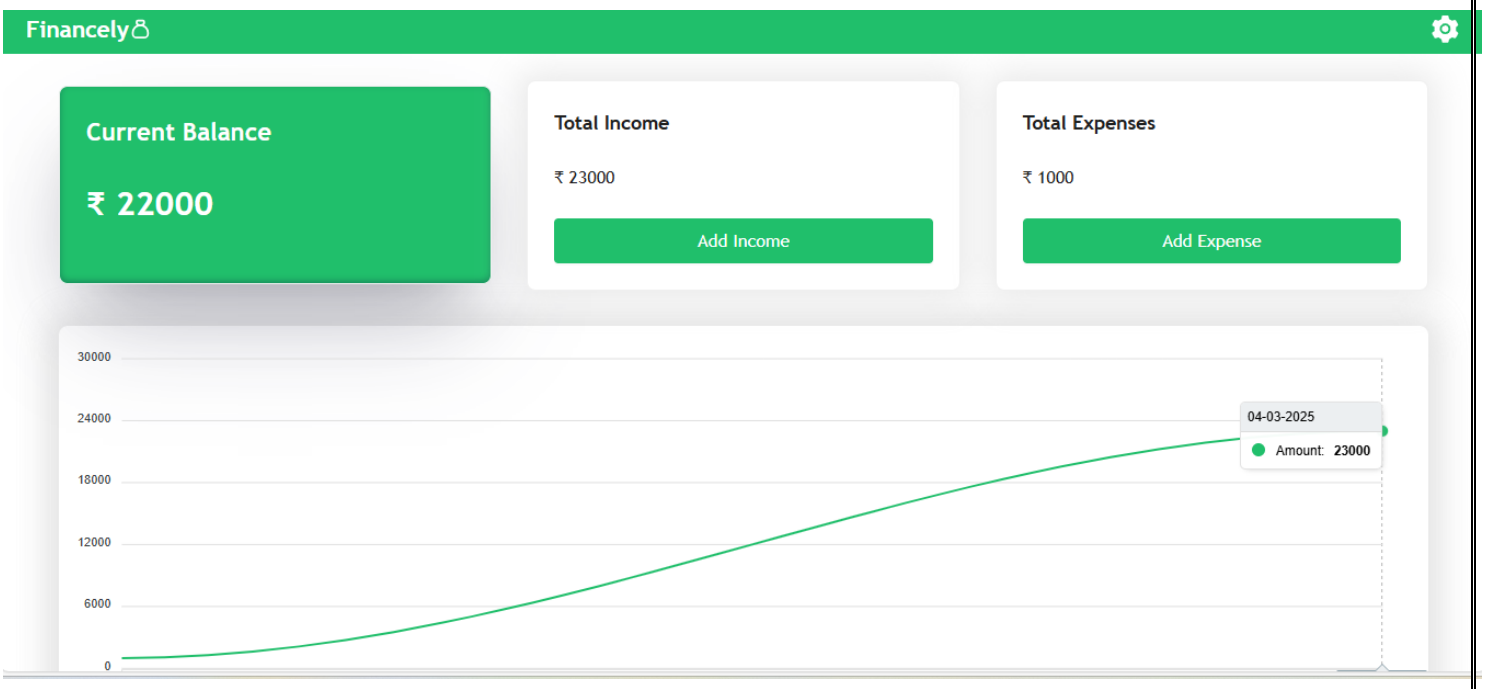
Sign up with Google

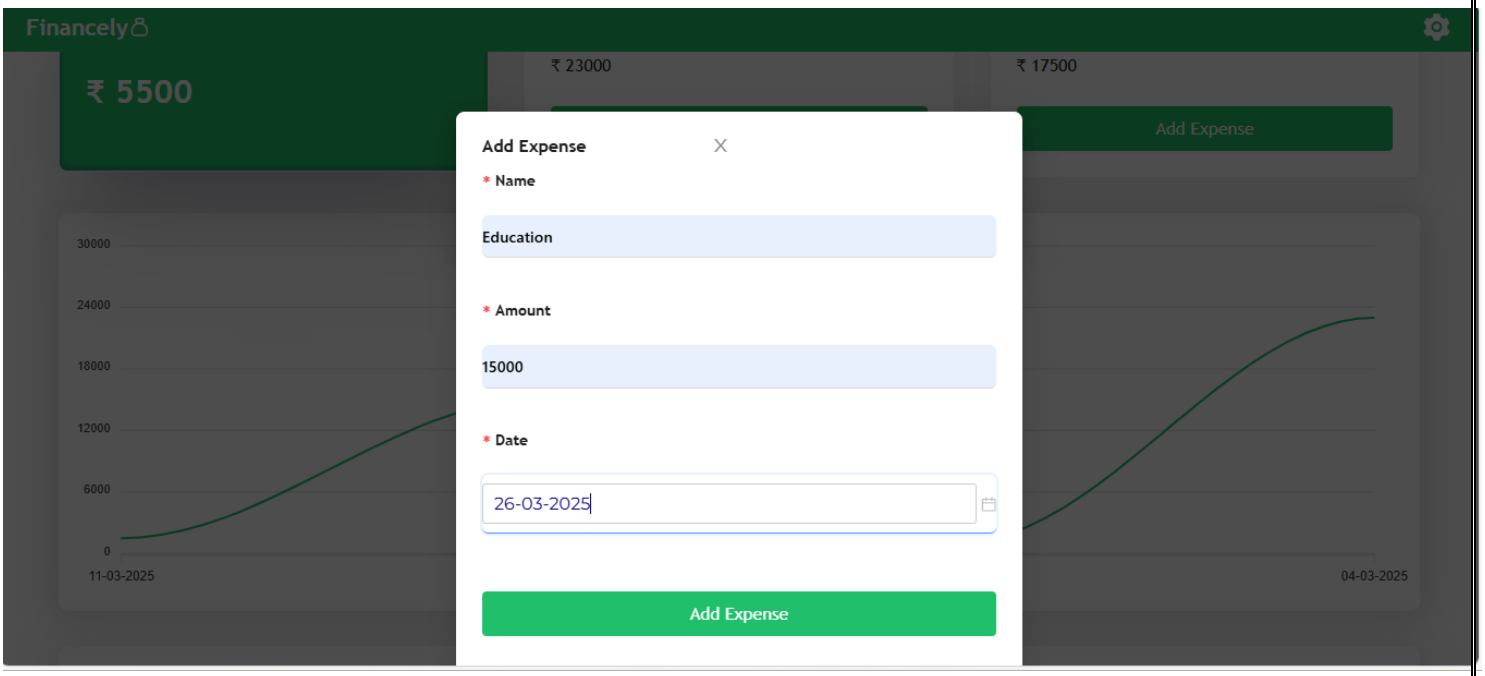
Already have an account? Click here

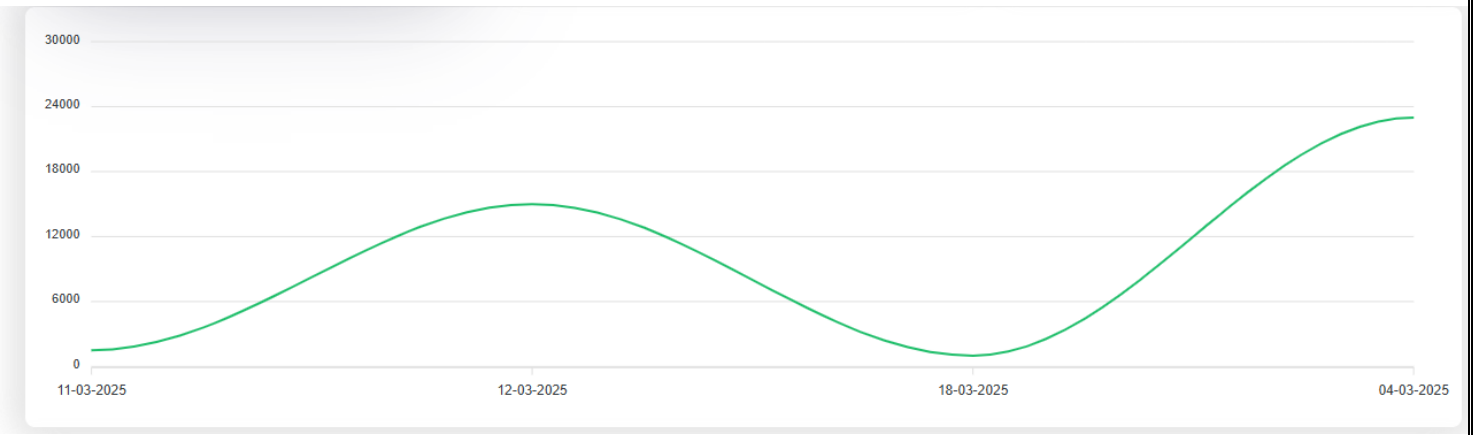

Financely
Login up on Fanancley
Enter your email
Password
Login
or
Login with Google
Don't have an account? Click here


Financely
Login up on Financley
Enter your email
Password
Login
or
Login with Google
Don't have an account? Click here


Financely
Current Balance
₹ 0
Total Income
₹ 0
Add Income
Total Expenses
₹ 0
Add Expense
No Transactions Avalaible


Financely
Current Balance
₹ 22000
Total Income
₹ 23000
Add Income
Total Expenses
₹ 1000
Add Expense
30000
24000
18000
12000
6000
0
04-03-2025
Amount: 23000


Financely
₹ 5500
₹ 23000
₹ 17500
Add Expense
Add Expense
* Name
Education
* Amount
15000
* Date
26-03-2025
Add Expense
30000
24000
18000
12000
6000
0
11-03-2025
04-03-2025


30000
24000
18000
12000
6000
0
11-03-2025
12-03-2025
18-03-2025
04-03-2025

My Transactions

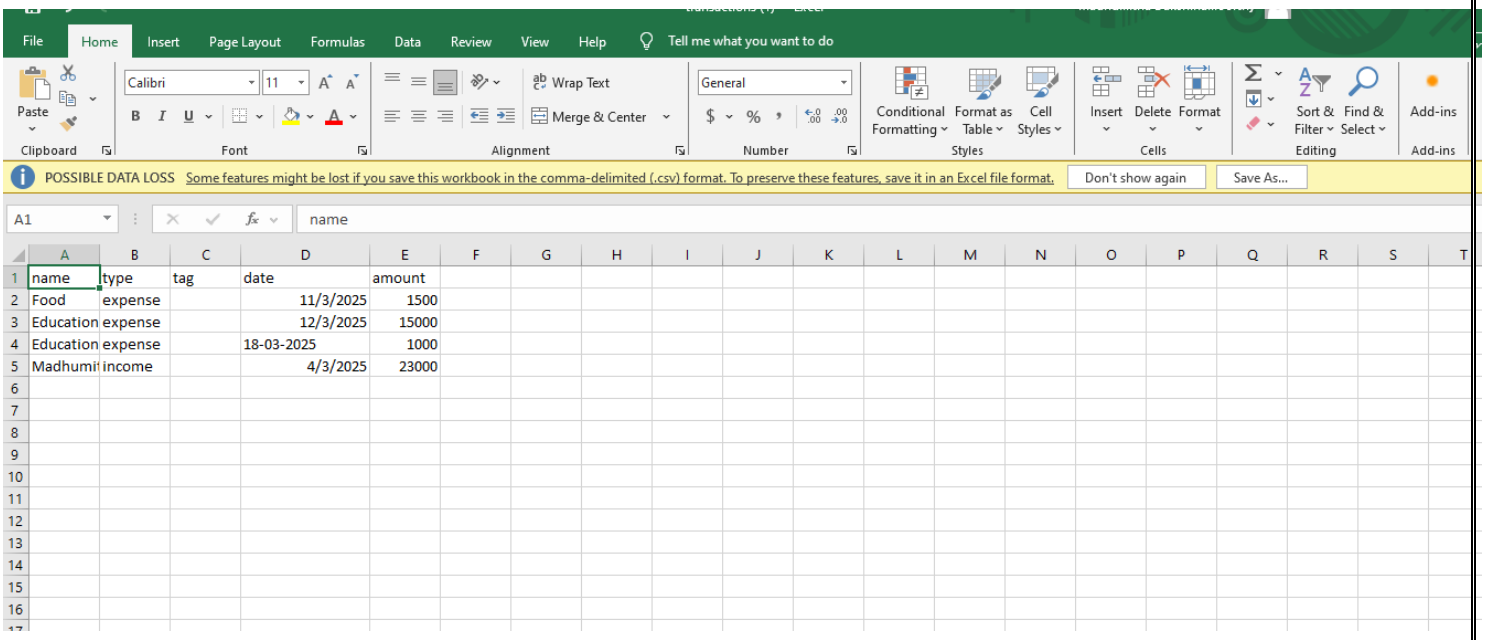
Search by name

All

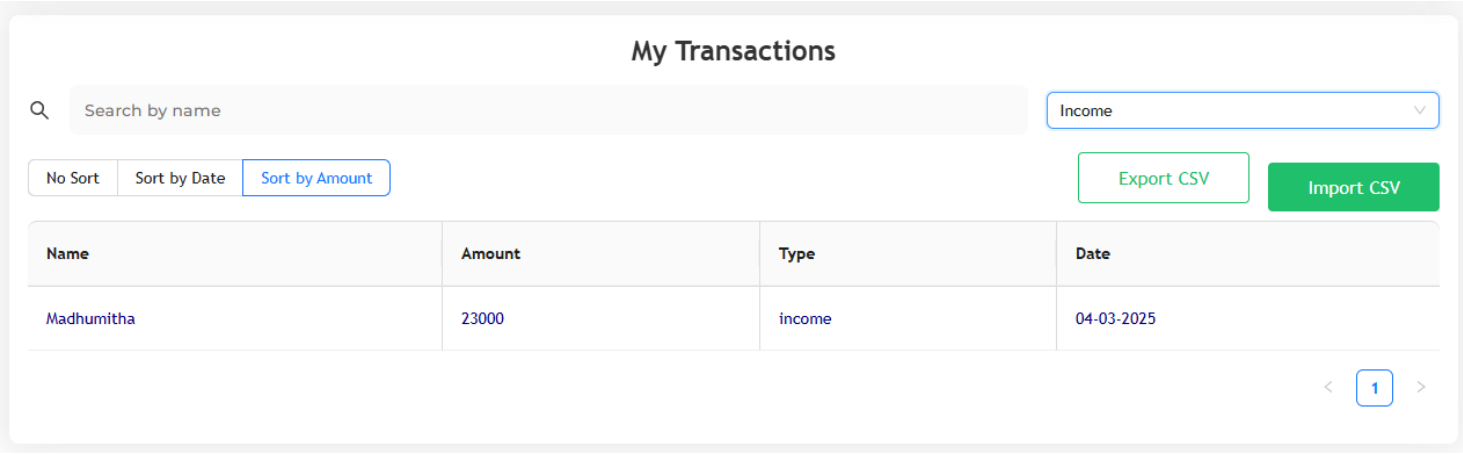
No Sort | Sort by Date | Sort by Amount

Export CSV | Import CSV

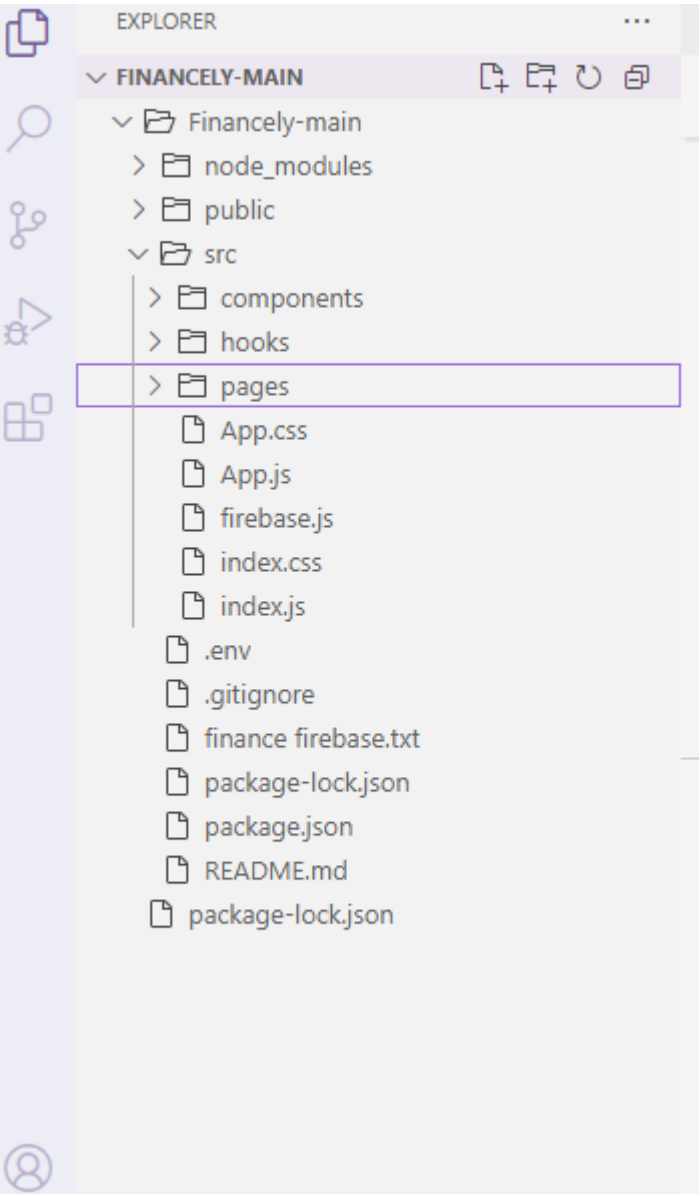
Name	Amount	Type	Date
Food	1500	expense	11-03-2025
Education	15000	expense	12-03-2025
Education	1000	expense	18-03-2025



	A	B	C	D	E
1	name	type	tag	date	amount
2	Food	expense		11/3/2025	1500
3	Education	expense		12/3/2025	15000
4	Education	expense		18-03-2025	1000
5	Madhumit	income		4/3/2025	23000


My Transactions
Search by name
Income
No Sort
Sort by Date
Sort by Amount
Export CSV
Import CSV
Name
Amount
Type
Date
Madhumitha
23000
income
04-03-2025
1

Structure:


EXPLORER
FINANCELY-MAIN
Financely-main
node_modules
public
src
components
hooks
pages
App.css
App.js
firebase.js
index.css
index.js
.env
.gitignore
finance firebase.txt
package-lock.json
package.json
README.md
package-lock.json

App.js

```
// import "./App.css";
import Header from "./components/Header";
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Home from "./pages/Home";
import Signup from "./pages/Signup";
import Dashboard from "./pages/Dashboard";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

function App() {
  return (
    <div className="App">
      <ToastContainer
        position="top-right"
        autoClose={3000}
        hideProgressBar={false}
        newestOnTop={false}
        closeOnClick
        rtl={false}
```

```
        pauseOnFocusLoss
        draggable
        pauseOnHover
        theme="light"
      />
      <Router>
        <Header />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/signup" element={<Signup />} />
          <Route path="/dashboard" element={<Dashboard />} />
        </Routes>
      </Router>
    </div>
  );
}

export default App;
```

App.css

```
.wrapper {
    display: flex;
```

```
    justify-content: center;
    align-items: center;
    width: 100vw;
    height: 90vh;
}
```

Firebase.js

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAuth, GoogleAuthProvider } from "firebase/auth";
import { getFirestore, doc, setDoc } from "firebase/firestore";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket:
process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
```

```
  messagingSenderId:
process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_APP_ID
};
// Initialize Firebase
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
const auth = getAuth(app);
const provider = new GoogleAuthProvider();

export { db, auth, provider, doc, setDoc };
```

Index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
```

```
    <App />
  </React.StrictMode>
);
```

Index.css

```
@import
url('https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100;
0,200;0,300;0,400;0,500;0,700;0,800;1,600;1,800&display=swap');


* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  text-decoration: none;
}

html {
  font-family: 'Montserrat', sans-serif;
  --webkit-font-smoothing: antialiased;
  --moz-osx-font-smoothing: grayscale;
  scroll-behavior: smooth;
```

```
}

::selection {
  background: var(--primary-purple);
  color: var(--white);
}

body {
  overflow-x: hidden;
}

input {
  font-family: 'Montserrat', sans-serif;
}

/* reusable code here */
:root {
  /* --theme: #2978ff; */
  --primary-green: #069067;
  --primary-purple: #6842EF;
  --primary-purple-shade: #8161f4;
```

```
  --black: #000;
  --white: #fff;
  --shadow-color: 0px 0px 30px 8px rgba(227, 227, 227, 0.75);
}

.container {
  max-width: 1250px;
  margin: 0 auto;
}

/* char styling  */
.chart {
  width: 95vw;
  margin-block: 2rem;
  box-shadow: var(--shadow-color);
  border-radius: .5rem;
  display: flex;
  justify-content: center;
  align-items: center;
}
```

```
.chart h2 {
  margin-top: 1rem;
  /* text-align: center; */
  opacity: .4;
}

.chart img {
  width: 50%;

}

.no-transaction {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

.line-chart {
  width: 95vw;
  box-shadow: var(--shadow-color);
```

```
  border-radius: .5rem;
}
```

.env

```
REACT_APP_FIREBASE_API_KEY=AIzaSyC0qJoKwr3dB6rhE9lqL-_JbWsOEUaUnks
REACT_APP_FIREBASE_AUTH_DOMAIN=finance-4525.firebaseapp.com
REACT_APP_FIREBASE_PROJECT_ID=finance-4525
REACT_APP_FIREBASE_STORAGE_BUCKET=finance-4525.firebasestorage.app
REACT_APP_FIREBASE_MESSAGING_SENDER_ID=47390121781
REACT_APP_FIREBASE_APP_ID=1:47390121781:web:d25615ef90427731eb767c
```

Dashboard.js

```
import React, { useEffect, useState } from "react";
import Cards from "../components/Cards";
import Modal from "antd/es/modal/Modal";
import AddExpense from "../components/Modals/addExpense";
import AddIncome from "../components/Modals/addIncome";
import { toast } from "react-toastify";
import { auth, db } from "../firebase";
import { addDoc, collection, getDocs, query } from "firebase/firestore";
import { useAuthState } from "react-firebase-hooks/auth";
import TransactionsTable from "../components/TransactionsTable";
import ChartComponent from "../components/Charts";
```

```
const Dashboard = () => {
  const [user] = useAuthState(auth);
  const [isExpenseModalVisible, setIsExpenseModalVisible] = useState(false);
  const [isIncomeModalVisible, setIsIncomeModalVisible] = useState(false);
  // all trasactions storing this array after that fetching into doc
  const [transactions, setTransactions] = useState([]);
  const [income, setIncome] = useState(0);
  const [expense, setExpense] = useState(0);
  const [currentBalance, setCurrentBalance] = useState(0);

  const showExpenseModal = () => {
    setIsExpenseModalVisible(true);
  };

  const showIncomeModal = () => {
    setIsIncomeModalVisible(true);
  };

  const handleExpenseCancel = () => {
    setIsExpenseModalVisible(false);
  };
```

```
const handleIncomeCancel = () => {
  setIsIncomeModalVisible(false);
};

// Adding a income and expense on the firebase new collection in diffrent uid
const onFinish = (values, type) => {
  const newTransaction = {
    type: type,
    date: values.date.format("DD-MM-YYYY"),
    amount: parseFloat(values.amount),
    name: values.name,
  };
  setTransactions([...transactions, newTransaction]);
  setIsExpenseModalVisible(false);
  setIsIncomeModalVisible(false);
  addTransaction(newTransaction);
};

const addTransaction = async (transaction, many) => {
  try {
    const docRef = await addDoc(
      collection(db, `users/${user.uid}/transactions`),
      transaction
```

```
    );
    if (!many) toast.success("Transaction Added!");
    // adding new transaction after that previous transactions
    setTransactions([...transactions, transaction]);
    calculateBalance();
    fetchTransactions();
  } catch (err) {
    if (!many) toast.error("Couldn't add transaction");
  }
};

const fetchTransactions = async () => {
  if (user) {
    const dataRef = query(collection(db, `users/${user.uid}/transactions`));
    const querySnapshot = await getDocs(dataRef);
    let transactionArray = [];
    querySnapshot.forEach((doc) => {
      transactionArray.push({ ...doc.data(), id: doc.id });
    });
    setTransactions(transactionArray);
    toast.success("Transaction Fetched!");
  }
};
```

```
useEffect(() => {
  // get all the docs from collections
  fetchTransactions();
}, [user]);

useEffect(() => {
  calculateBalance();
}, [transactions]);

// Calculate the inital FaBalanceScale, income and expenses
const calculateBalance = () => {
  let totalIncome = 0;
  let totalExpense = 0;

  transactions.forEach((transaction) => {
    if (transaction.type === "income") {
      totalIncome += parseFloat(transaction.amount);
    } else {
      totalExpense += parseFloat(transaction.amount);
    }
  });
  setIncome(totalIncome);
```

```
  setExpense(totalExpense);
  setCurrentBalance(totalIncome - totalExpense);
};

return (
  <div>
    <Cards
      showExpenseModal={showExpenseModal}
      showIncomeModal={showIncomeModal}
      income={income}
      expense={expense}
      currentBalance={currentBalance}
    />
    <Modal open={isIncomeModalVisible} onCancel={handleIncomeCancel}>
      Income
    </Modal>
    <Modal open={isExpenseModalVisible} onCancel={handleExpenseCancel}>
      Expense
    </Modal>
    <AddExpense
      isExpenseModalVisible={isExpenseModalVisible}
      handleExpenseCancel={handleExpenseCancel}
      onFinish={onFinish}
```

```
/>
<AddIncome
  isIncomeModalVisible={isIncomeModalVisible}
  handleIncomeCancel={handleIncomeCancel}
  onFinish={onFinish}
/>
<div className="chart container">
  {transactions.length !== 0 ? (
    <div className="line-chart">
      <ChartComponent transactions={transactions} />
    </div>
  ) : (
    <div className="no-transaction">
      <h2>No Transactions Avalaible</h2>
      <img
        src={process.env.PUBLIC_URL + "/coin.gif"}
        alt="No-transaction-img"
      />
    </div>
  )}
</div>
<TransactionsTable
  transactions={transactions}
```

```
        addTransaction={addTransaction}
        fetchTransactions={fetchTransactions}
      />
    </div>
  );
};

export default Dashboard;
```

Home.js

```
import React, { useEffect } from "react";
import { useNavigate } from "react-router-dom";
import "./Home.css"; // Home-specific styles

const Home = () => {
  const navigate = useNavigate();

  useEffect(() => {
    // Apply styles when Home.js is loaded
    document.body.style.scrollBehavior = "smooth";
    document.body.style.backgroundColor = "#0b0081";

    return () => {
```

```
    // Reset styles when leaving Home.js
    document.body.style.scrollBehavior = "";
    document.body.style.backgroundColor = "";
  };
}, []);
return (
  <div>
    {/* Navbar */}
    <nav>
      <div className="navbar1">
        <a href="#" className="logo">
          <i className="fas fa-chart-line"></i> Finance Tracker
        </a>
        <ul className="nav-links">
          <li><a href="#">Home</a></li>
          <li><a href="#tracker">Use Tracker</a></li>
          <li><a href="#how-it-works">How it Works</a></li>
        </ul>
        {/* <div className="buttons">
          <button className="btn-head" onClick={() => navigate("/login")}>
            <i className="fas fa-sign-in-alt"></i> Log In
          </button>
          <button className="btn-head" onClick={() => navigate("/signup")}>
```

```
          <i className="fas fa-user-plus"></i> Sign Up
        </button>
      </div> */}
    </div>
  </nav>

  {/* Landing Page Content */}
  <section className="section-box">
    <div className="main-landing">
      <div className="content">
        <h1 className="under-welcome-h1">Take Control of Your Finances</h1>
        <p className="under-welcome-p">
          Track your income and expenses effortlessly. Manage your budget, set
          financial goals, and make smarter decisions for a more secure future.
        </p>
        <button className="under-welcome-btn" onClick={() =>
navigate("/signup")}>
          Sign Up Now
        </button>
      </div>
      <div className="under-welcome-image">
      {/* <img src="/coin.gif" alt="Power Finance" /> */}
      <img src="/makeme.png" alt="Power Finance" />
```

```
    </div>
  </div>
</section>

{/* Scroll Down Arrow */}
<div className="arrow">
  <a href="#tracker" className="scroll-link">
    <span></span>
  </a>
</div>

{/* Tracker Section */}
<section className="main-content" id="tracker">
  <div className="head-text">
    <h1>Access Your Personal Finance Tracker</h1>
  </div>
  <div className="tracker-part">
    <div className="tracker-balance">
      Current Balance: <span id="balance" className="current-balance"></span>
    </div>
    <div className="currency-filter">
      <div className="filter-left">
        <label htmlFor="currency">Select Currency:</label>
```

```
    <select id="currency">
      <option value="USD">USD</option>
      <option value="EUR">EUR</option>
      <option value="INR" selected>INR</option>
    </select>
  </div>
  <div className="filter-right">
    <label htmlFor="date">Choose Date:</label>
    <input type="date" id="date" />
  </div>
</div>
<div className="transaction-form">
  <input type="text" id="description" placeholder="Description" />
  <input type="number" id="amount" placeholder="Amount" />
  <select id="type">
    <option value="income">Income</option>
    <option value="expense">Expense</option>
  </select>
  <button>Add Transaction</button>
  <button>Export</button>
</div>
<div className="table-part" id="table-part">
  <table id="transaction-table">
```

```
        <thead>
          <tr>
            <th>Date</th>
            <th>Description</th>
            <th>Amount</th>
            <th>Type</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          {/* Transactions will be inserted dynamically */}
        </tbody>
      </table>
    </div>
  </div>
</section>

{/* How It Works Section */}
<section className="how-it-works" id="how-it-works">
  <div className="head-text">
    <h1>How it Works</h1>
  </div>
  <div className="working-content">
```

```
        <div className="left-working">
          <h5 className="working-head">
            Experience the power of
            <span className="text-lightBlue"> Personal Finance Tracker</span>
          </h5>
          <ul className="ul-items">
            <li><i className="fas fa-check"></i> Track effortlessly</li>
            <li><i className="fas fa-check"></i> Control transactions</li>
            <li><i className="fas fa-check"></i> Customize currencies</li>
            <li><i className="fas fa-check"></i> Edit with ease</li>
            <li><i className="fas fa-check"></i> Real-time balance</li>
            <li><i className="fas fa-check"></i> Secure data protection</li>
          </ul>
          <button className="sign-up-button-btn" onClick={() =>
navigate("/signup")}>
            Sign Up Now
          </button>
        </div>
        <div className="right-working-image">
          <img src="/how-it-works.png" alt="Power Finance" />
        </div>
      </div>
    </section>
```

```
      {/* Footer */}
      <footer>
        <div className="footer">
          <p>© 2025 Finance Tracker. All rights reserved.</p>
        </div>
      </footer>
    </div>
  );
};

export default Home;
```

Home.cs

```
import React, { useEffect } from "react";
import { useNavigate } from "react-router-dom";
import "./Home.css"; // Home-specific styles

const Home = () => {
  const navigate = useNavigate();

  useEffect(() => {
    // Apply styles when Home.js is loaded
```

```
  document.body.style.scrollBehavior = "smooth";
  document.body.style.backgroundColor = "#0b0081";

  return () => {
    // Reset styles when leaving Home.js
    document.body.style.scrollBehavior = "";
    document.body.style.backgroundColor = "";
  };
}, []);
return (
  <div>
    {/* Navbar */}
    <nav>
      <div className="navbar1">
        <a href="#" className="logo">
          <i className="fas fa-chart-line"></i> Finance Tracker
        </a>
        <ul className="nav-links">
          <li><a href="#">Home</a></li>
          <li><a href="#tracker">Use Tracker</a></li>
          <li><a href="#how-it-works">How it Works</a></li>
        </ul>
        {/* <div className="buttons">
```

```
        <button className="btn-head" onClick={() => navigate("/login")}>
          <i className="fas fa-sign-in-alt"></i> Log In
        </button>
        <button className="btn-head" onClick={() => navigate("/signup")}>
          <i className="fas fa-user-plus"></i> Sign Up
        </button>
      </div> */}
    </div>
  </nav>

  {/* Landing Page Content */}
  <section className="section-box">
    <div className="main-landing">
      <div className="content">
        <h1 className="under-welcome-h1">Take Control of Your Finances</h1>
        <p className="under-welcome-p">
          Track your income and expenses effortlessly. Manage your budget, set
          financial goals, and make smarter decisions for a more secure future.
        </p>
        <button className="under-welcome-btn" onClick={() =>
navigate("/signup")}>
          Sign Up Now
        </button>
```

```
    </div>
    <div className="under-welcome-image">
    {/* <img src="/coin.gif" alt="Power Finance" /> */}
    <img src="/makeme.png" alt="Power Finance" />
    </div>
  </div>
</section>

{/* Scroll Down Arrow */}
<div className="arrow">
  <a href="#tracker" className="scroll-link">
    <span></span>
  </a>
</div>

{/* Tracker Section */}
<section className="main-content" id="tracker">
  <div className="head-text">
    <h1>Access Your Personal Finance Tracker</h1>
  </div>
  <div className="tracker-part">
    <div className="tracker-balance">
      Current Balance: <span id="balance" className="current-balance"></span>
```

```
</div>
<div className="currency-filter">
  <div className="filter-left">
    <label htmlFor="currency">Select Currency:</label>
    <select id="currency">
      <option value="USD">USD</option>
      <option value="EUR">EUR</option>
      <option value="INR" selected>INR</option>
    </select>
  </div>
  <div className="filter-right">
    <label htmlFor="date">Choose Date:</label>
    <input type="date" id="date" />
  </div>
</div>
<div className="transaction-form">
  <input type="text" id="description" placeholder="Description" />
  <input type="number" id="amount" placeholder="Amount" />
  <select id="type">
    <option value="income">Income</option>
    <option value="expense">Expense</option>
  </select>
  <button>Add Transaction</button>
```

```
      <button>Export</button>
    </div>
    <div className="table-part" id="table-part">
      <table id="transaction-table">
        <thead>
          <tr>
            <th>Date</th>
            <th>Description</th>
            <th>Amount</th>
            <th>Type</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          {/* Transactions will be inserted dynamically */}
        </tbody>
      </table>
    </div>
  </div>
</section>

{/* How It Works Section */}
<section className="how-it-works" id="how-it-works">
```

```
      <div className="head-text">
        <h1>How it Works</h1>
      </div>
      <div className="working-content">
        <div className="left-working">
          <h5 className="working-head">
            Experience the power of
            <span className="text-lightBlue"> Personal Finance Tracker</span>
          </h5>
          <ul className="ul-items">
            <li><i className="fas fa-check"></i> Track effortlessly</li>
            <li><i className="fas fa-check"></i> Control transactions</li>
            <li><i className="fas fa-check"></i> Customize currencies</li>
            <li><i className="fas fa-check"></i> Edit with ease</li>
            <li><i className="fas fa-check"></i> Real-time balance</li>
            <li><i className="fas fa-check"></i> Secure data protection</li>
          </ul>
          <button className="sign-up-button-btn" onClick={() =>
navigate("/signup")}>
            Sign Up Now
          </button>
        </div>
        <div className="right-working-image">
```

```
          <img src="/how-it-works.png" alt="Power Finance" />
        </div>
      </div>
    </section>

    {/* Footer */}
    <footer>
      <div className="footer">
        <p>© 2025 Finance Tracker. All rights reserved.</p>
      </div>
    </footer>
  </div>
 );
};

export default Home;
```

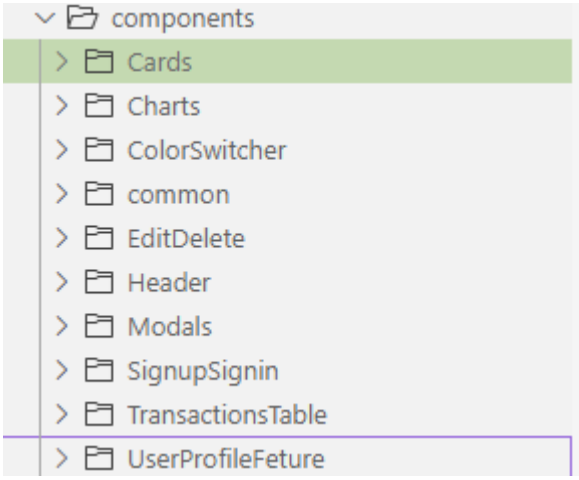
Signup.js

```
import React from 'react'
import "../App.css";
import SignupSignin from "../components/SignupSignin"

const Signup = () => {
```

```
  return (
    <div className='wrapper'>
      <SignupSignin />
    </div>
  )
}

export default Signup;
```



Index.js

```
import "./styles.css";
import { Button, Card, Row } from "antd";

const Cards = ({
  showExpenseModal,
  showIncomeModal,
```

```
  income,
  expense,
  currentBalance,
}) => {
  return (
    <div>
      <Row className="card-row container">
        <Card className="mycard current-balance">
          <h2 className="title">Current Balance</h2>
          <p>₹ {currentBalance}</p>
        </Card>
        <Card className="mycard">
          <h2 className="title">Total Income</h2>
          <p>₹ {income}</p>
          <Button className="btn reset-balance-btn" onClick={showIncomeModal}>
            Add Income
          </Button>
        </Card>
        <Card className="mycard">
          <h2 className="title">Total Expenses</h2>
          <p>₹ {expense}</p>
          <Button className="btn reset-balance-btn" onClick={showExpenseModal}>
            Add Expense
```

```
          </Button>
        </Card>
      </Row>
    </div>
  );
};

export default Cards;
```

Style.css

```
.mycard {
    box-shadow: var(--shadow-color);
    min-width: 19rem;
    font-family: 'Montserrat', sans-serif;
    flex: 1;
}

.current-balance {
    color: var(--white);
    background-color: var(--primary-purple);
    /* background-image: linear-gradient(315deg, #6842EF 0%, #8161f4 74%); */
    box-shadow: rgba(50, 50, 93, 0.25) 0px 50px 100px -20px, rgba(0, 0, 0, 0.3) 0px 30px 60px -30px, rgba(10, 37, 64, 0.35) 0px -2px 6px 0px inset !important;
```

```
}

.current-balance h2 {
    font-size: 1.4rem !important;
}

.current-balance p {
    font-size: 1.9rem !important;
    font-weight: 600 !important;
    text-shadow: var(--shadow-color) !important;
}

.title {
    font-size: 1.1rem;
    text-align: left;
}

.mycard p {
    font-size: 1rem;
    font-weight: 500;
    margin-block: 1.5rem;
}
```

```
.card-row {
    display: flex;
    justify-content: space-between;
    align-items: center;
    flex-wrap: wrap;
    gap: 2rem;
    width: 95vw;
    margin: 1.5rem auto;
}

.reset-balance-btn {
    padding: .5rem !important;
}

.reset-balance-btn:hover {
    letter-spacing: 1px;
    color: var(--white) !important;
}
```